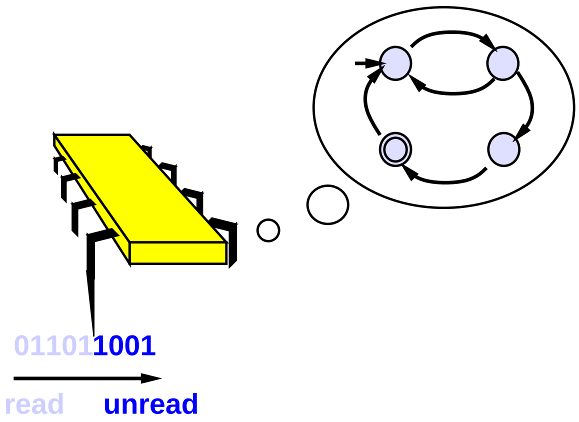
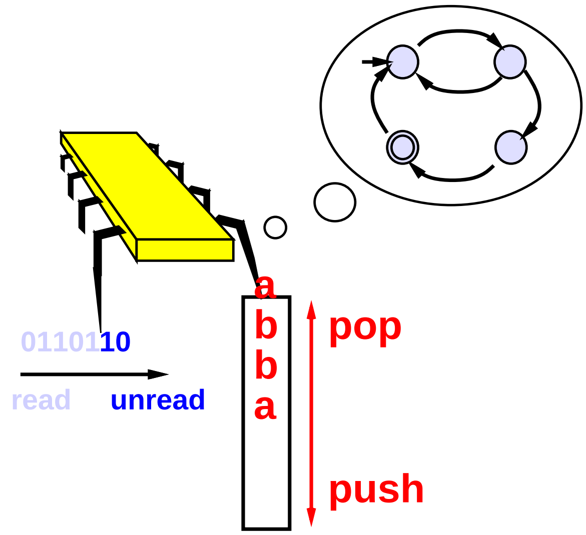
A Finite Automaton



A Pushdown Automaton



A Pushdown Automaton

- can push symbols onto the stack
- can pop them (read them back) later
- stack is potentially unbounded

An Example

Recall that $0^n 1^n$ not regular.

Consider the following PDA:

- read input symbols
- for each 0, push it on the stack
- as soon as a 1 is seen, pop a 0 for each 1 read
- accept if stack empty when last symbol read.
- reject if stack non-empty, if input symbol exist, if 0 read after 1, etc.

Non-Determinism

PDA may be non-deterministic.

PDA *must* be non-deterministic.

Unlike finite automata, non-determinism adds power.

Formal Definition

Use a different alphabet for inputs Σ and for stack Γ.

Transition function looks different.

From:

- current state: Q
- next input, if any: Σ_ε
- stack symbol symbol popped, if any: Γ_ε

To:

- new state: Q
- stack symbol pushed, if any: Γ_ε
- non-determinism: $\mathcal{P}(\cdots)$

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$$

Formal Definitions

A *pushdown automaton* (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- Q is a finite set called the states,
- Σ is a finite set called the input alphabet,
- Γ is a finite set called the *stack alphabet*,
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q)$ is the *transition function*,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states.

Conventions

Question: When is the stack empty?

- start by pushing \$ onto stack
- if you see it again, stack is empty.

When is input string exhausted?

- doesn't matter
- accepting state accepts only if inputs exhausted!

Notation

Transition

$$a, b \to c$$

means

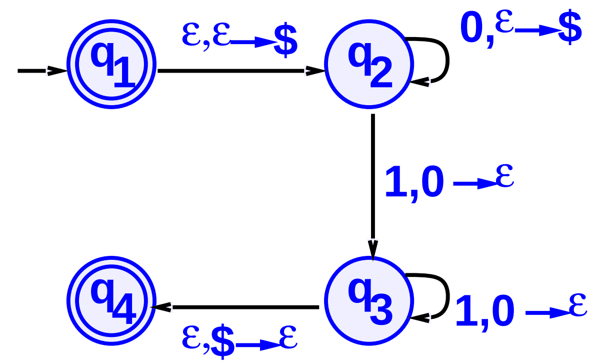
- read a from input
- pop b from stack
- push c onto stack

Meaning of ε transitions:

- if $a = \varepsilon$, don't read inputs
- if $b = \varepsilon$, don't pop any symbols
- if $c = \varepsilon$, don't push any symbols

Example

The PDA



accepts

$$\{0^n 1^n | n \geq 1\}.$$

Another Example

A PDA that accepts

$$\left\{a^i b^j c^k | i, j, k > 0 \text{ and } i = j \text{ or } i = k\right\}$$

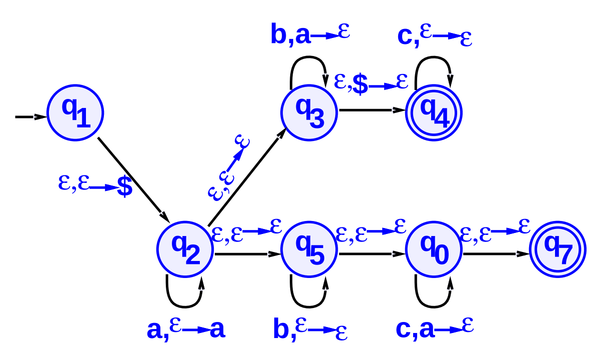
Informally:

- read and push a's
- either pop and match with b's
- or else pop and match with c's
- non-deterministic choice!

Note: non-determinism essential here!

Unlike finite automata, non-determinism adds power

Another Example



A PDA that accepts

$$\left\{a^i b^j c^k | i, j, k > 0 \text{ and } i = j \text{ or } i = k\right\}$$

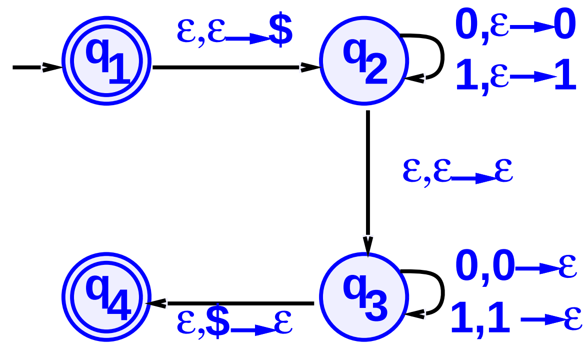
Yet Another Example

A *palindrome* has the form $ww^{\mathcal{R}}$.

- "Madam I'm Adam"
- "Dennis and Edna sinned"
- "Red rum, sir, is murder"
- "In girum imus nocte et consumimur igni"

Yet Another Example

This PDA



accepts binary palindromes.

Theorem

Theorem: A language is context free if and only if some pushdown automaton accepts it.

This time, both the "if" part and the "only if" part are interesting.

If Part

Theorem: If a language is context free, then some pushdown automaton accepts it.

- Let A be a context-free language.
- We know A has a context-free grammar G.
- on input w, the PDA P figures out if there is a a derivation of w using G.

Question: How does P figure out which substitution to make?

Answer: It guesses.

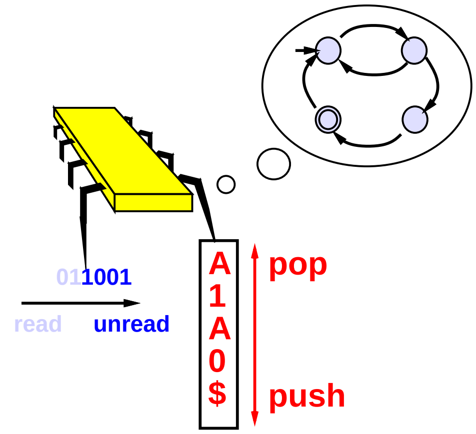
CFL Implies PDA

Informally:

- P pushes start variable S on stack
- keeps making substitutions
- when only terminals remain ...
- tests whether derived string equals input

CFL Implies PDA

Where do we keep the intermediate string?



intermediate string: 01A1A0

- can't put it all on the stack
- only symbols starting with first variable on stack

CFL Implies PDA

Informal description:

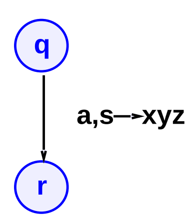
- push $S\$$ on stack
- if top of stack is variable A, non-deterministically select rule and substitute.
- if top of stack is terminal a read next input and compare. If they differ, reject.
- if top of stack is \$, enter accept state. (Really accepts only if input has all been read!).

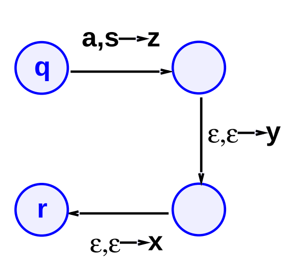
CFL Implies PDA

Need shorthand to push entire string onto stack.

$$(r, w) \in \delta(q, a, s)$$

Easy to do by introducing intermediate states.





CFL Implies PDA

States of P are

- start state q_s
- accept state q_a
- loop state q_ℓ
- E states needed for shorthand

Transition Function

Initialize stack

$$\delta(q_s, \varepsilon, \varepsilon) = \{q_\ell, S\$\}$$

Top of stack is variable

$$\delta(q_\ell, \varepsilon, A) = \{(q_\ell, w) | \text{ where } A \to w \text{ is a rule } \}$$

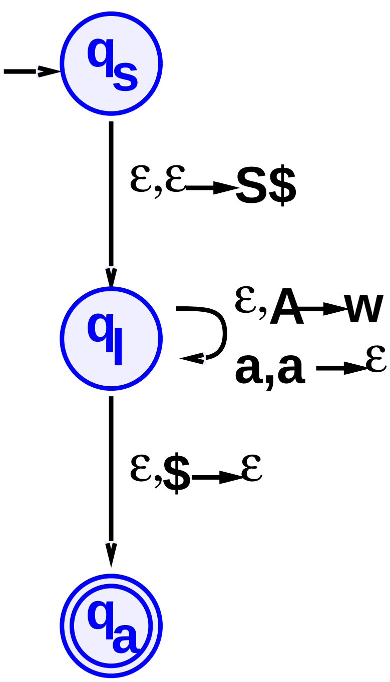
Top of stack is terminal

$$\delta(q_\ell, a, a) = \{(q_\ell, \varepsilon)\}$$

End of Stack

$$\delta(q_\ell, \varepsilon, \$) = \{(q_a, \varepsilon)\}$$

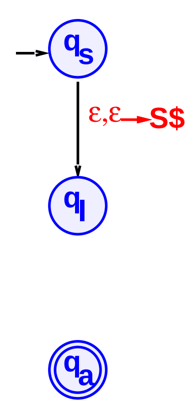
Transition Function



Example

$$\begin{aligned} S &\to aTb|b \\ T &\to Ta|\varepsilon \end{aligned}$$

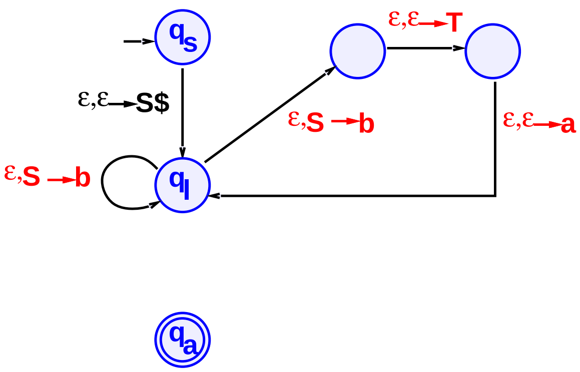
Initialization:



Example

$$
\begin{aligned}
S &\rightarrow aTb|b \\
T &\rightarrow Ta|\varepsilon
\end{aligned}
$$

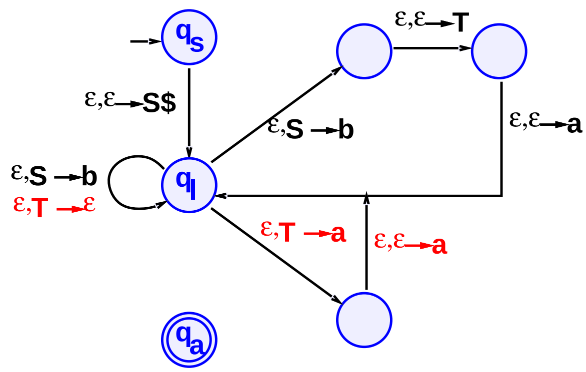
Rules for S



Example

$$
\begin{aligned}
S &\rightarrow aTb|b \\
T &\rightarrow Ta|\varepsilon
\end{aligned}
$$

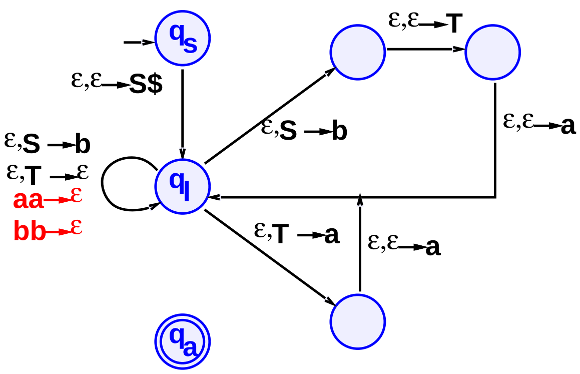
Rules for T



Example

$$
\begin{aligned}
S &\rightarrow aTb|b \\
T &\rightarrow Ta|\varepsilon
\end{aligned}
$$

Rules for terminals



Example

$$
\begin{aligned}
S &\rightarrow aTb|b \\
T &\rightarrow Ta|\varepsilon
\end{aligned}
$$

Termination: